Generation and Prioritization of test sequences using UML activity diagram

A Thesis submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology in Computer Science and Engineering

BY

Ashu Mishra

(110CS0134)

Under the Guidance of

Prof. D. P. Mohapatra

Associate Professor,

Department of Computer Science & Engg.

NIT Rourkela

Department of Computer Science and Engineering
National Institute of Technology,
Rourkela-769008
May-2014

Certificate

May 2014

This is to certify that the project entitled **"Generation and Prioritization of test sequences using UML activity diagram"** submitted by Ashu Mishra (110CS0134) in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Computer Science & Engineering at NIT Rourkela is an authentic work carried out by him under my guidance.

To the best of my knowledge the matter embodied in the project has not been submitted to any University/Institute for the award of any B.Tech degree.

Prof. D. P. Mohapatra
Associate Professor,
Dept. of Computer Science & Engg.
NIT Rourkela.

Declaration

I hereby declare that all the work contained in this report is my own work unless otherwise acknowledged. Also, all of my work has not been previously submitted for any academic degree. All sources of quoted information have been acknowledged by means of appropriate references.

Ashu Mishra

NIT Rourkela

Acknowledgement

The project work would not have been finished without mentioning those who have provided their knowledge and guidance. I would like to express my gratitude to Prof. D. P. Mohapatra for his constant advice and guidance throughout my project work. As my supervisor, he has constantly kept me motivated to stay focused towards achieving the goal. His observations and wisdom helped me to establish an overall direction for the research and to make an in depth study. Without his useful advice and assistance it would not have been possible for me to complete this thesis. I am very much thankful to the faculty members of Department of Computer Science and Engineering, National Institute of Technology, Rourkela especially Mr. Vikas Panthi for his constant support and help during the project work.

Ashu Mishra

Abstract

Software testing is one of the most important part in software development life cycle. Generation of test sequences which is essential for software testing is one of the challenging task. Testing can be done either manually or automatically. Automated testing is better than manual testing as it reduces cost as well as time. UML activity diagram are used to show the workflow of dynamic and behavioural aspects of the software system. Test sequences can be generated using the UML activity diagram and different prioritization techniques can be applied to select the effective paths. In this project, first we generate test sequences using the activity diagram. Then, we prioritize test sequences using Ant colony optimization. We have implemented the proposed algorithm. We have compared the proposed prioritization technique with the existing prioritization technique. Experimental results shows that our result is better than that of the existing technique.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Producing software with higher quality is the basic objective of software testing. Software testing is conducted to find the faults which causes the software to fail. But also it is time consuming and expensive to perform. Software testing can be either manual or automatic. Automated testing is usually done using software testing tools. Automated software testing is normally more efficient than manual testing as it reduces time and cost amount.

Verity of software testing leads to standardization. Some major software testing process includes black box testing and white box testing. Basically software testing is a technique to validate and verify the program whether it meets the customer's need or not.

Black box testing is known for functional testing. It does not check how the program is implemented. Black box testing ensures functionality of a software (what it does) without having any knowledge about the internal structure.

White box testing tests all possible path that are available in a software. It checks the internal structure and working of a software. It verifies the quality of the source code.

Over some years, UML model-based testing has evolved to face challenges of large and complex industrial software systems. The use of UML specification for software development has led us to a new way of development. It also changed the testing process. UML has taken root in analysing and designing large systems. Moreover, it is a de-facto standard for both software industry as well as academics. UML analysis and design are

performed in many software projects. It is a standard modelling language for visually describing the structure and understanding the behaviours of system.

Unified modelling language is used to model dynamic and also static behaviour of a software. Activity, sequence and state diagram are used to represent the dynamic behaviour of the software whereas class, component and deployment diagrams are used to represent the static behaviour of the software. Activity diagram depicts the activities of the different objects of the software, so the operations can be realized in the design stage itself.

Generation of test sequences is very difficult. Proper algorithm must be implemented to generate required test sequence from the UML diagrams. Out of those test sequences, one or two will be selected for testing the software.

Ant Colony Optimization [6] is another approach for generation of test sequences. It is a probabilistic algorithm used to solve problems to generate best path from a graph. This algorithm is inspired by the behaviour of ants for finding their path in the colony of foods.

1.1 Motivation

Software testing takes a huge amount of time and effort of the total software development process. As the size of the software increases so does its complexity which increases the maintenance cost and time taken for testing and debugging. The input database of a software is so huge that it is very difficult to make a test sequence that can test all the defect of the software on one try. Proper testing requires detection of error and fault in the software. Test sequences can be used to detect those errors. Generation of test sequences from a software and also selection of the best test sequence out of them needs proper techniques.

Generating all test sequences might not be a good approach. Because as the size and complexity of the software increases, number of test sequences generated from it also increases. So we need a method to limit those test sequences so that it can reduce both time and cost for testing a software.

1.2 Objective

The objective of this project is to parse the UML diagram into a program and generate test sequences out of it. From those, test sequence having higher priority is selected for software testing. Ant Colony Optimization technique [6] is implemented and compared with the other algorithm. Modified version of ACO is proposed and compared with the existing algorithm.

Chapter 2

Basic Concepts

In this section we discuss some basic definitions and concepts.

A Test sequence is a path generated from the activity diagram. It consists of a set of nodes. If we traverse the activity diagram from Start node to End node, nodes encountered during that time are stored in some array. If we reach the end point using that algorithm, then the array is considered to be a test sequence. Test sequence gives the path on which testing should be done. It is our job to select the best test sequence from it so that the software will be tested more efficiently.

2.1 Unified Modelling language (UML)

UML, has become popular now-a-days as it is a de-facto standard in industry for modelling object-oriented systems. It was developed in early 1990s. Overall nine diagrams are used to observe a system. It was received as a de-facto standard for modelling software system by OMG in 1997.

Activity Diagram: It is the workflow of dynamic and behavioural aspect of the system. It is similar to the state chart diagram because activity diagram has states doing something. The diagram describes about the ordering of actions. It may be considered as the flow chart of the source code. Activity diagram describes the internal behaviour of a work. It also has the facility to show conditional and parallel activities. Edges denotes the transition from one activity to another. Here condition can be shown by using Decision node and parallelism can be denoted by Fork node. It has an Initial node, where the activity starts and a Final node

where the activity ends. Activity diagrams are read from top to bottom. Transactions are represented by arrows.

2.2 Ant Colony Optimization

It is an algorithm based on the behaviour of real ants. First ACO technique was proposed as Ant System [9] and was applied to the traveling salesman problem. Many variation and different algorithm has been proposed since then. It is basically the behaviour of ants and the pheromone trail left on the path by them whenever they move toward the food source. The trails can be sensed by other ants and they will decide which path to choose or not. ACO is a probabilistic technique that is applied to generate solutions for combinatorial optimization problems. The artificial ants in the algorithm are used to find the path in the activity diagram.

The main theme of ACO is that, in real world, Ants wander to find food and lay pheromone trails on their path. If other ant find this trail, then it would stop wandering and start following the trail in search of food. However, the trail of pheromone start to evaporate after sometime, so it loses its attraction strength. The more ant take time to travel from one end to another, the more the path loses its pheromone trail. So a shorter path have more possibility to have the stronger pheromone level. If no evaporation occur at all, then the path chosen by the first ant will be followed by other ants. Thus, if one ant choses a shortest path, other ants are more likely to follow it, and the usefulness of the path is spread to other ants. So all ants will follow a single pheromone trail left the first ant and keep its level strong. The knowledge of this theorem is used to solve traversal in graph.

Chapter 3

Related Work

Chandler et al. [10] proposed an approach to Generate test sequences from an UML model file of an activity diagram without generating any intermediate model. This technique elaborates on the use of XML meta-data and extract necessary data from it. XML meta-data interchange (XMI) code can be extracted from UML the activity diagram by using different software tools. Using this information about nodes and edges can be extracted. This data can be viewed as graph. From that test sequences can be easily generated. Sun et al. [11] proposed the TSGen tool to automatically generate the test sequences based on UML activity diagram. The diagram is processed to store its action and transition information. The information is then used to generate test sequences.

Shirole e al [2] suggested a concurrent queue search to generate test sequences with concurrency. Concurrent queue search is used to maintain random nature of concurrent task in an Activity diagram. They presented how to convert a sequence diagram into an activity diagram. Using the queues to store action nodes, decision nodes, fork nodes etc. they demonstrated the generation of test sequences using that queue. Their technique for generation of test sequences are more preferable to DFS and BFS search algorithm.

Farooq et al. [14], observed that he Activity Diagram model can be easily converted into a Coloured Petri Net. They proposed a technique that enables the automatic generation of test sequences according to a given coverage criteria from the execution of the Coloured Petri Nets model. They described two structural coverage criteria for activity diagram, Sequential and concurrent. The proposed technique was applied to an example to demonstrate its feasibility and the generated test sequences were evaluated against selected coverage criteria.

The technique can potentially be adapted to service oriented applications, workflows, and concurrent applications.

Ian Sommerville, stated some major methods of software testing that includes black box testing and white box testing.

Sangeeta et al [1]. Illustrated the prioritization of the test sequences using Genetic algorithm. They used FAN-IN and FAN-OUT technique to assign weights to different nodes. Using those weights they calculated the strength of each test sequences and accordingly prioritized them.

Praveen et al. [6] proposed an approach for generation of test sequences using Ant colony optimization. It consists of path generation using the behaviour of ants. They presented a simple algorithm with the help of an ant colony optimization (ACO) technique, for the optimal path identification by using the basic property and behaviour of the ants. The approach consists of certain collection of method and principles to find out all the useful or effective paths via ant ACO technique. The method concentrates on generation of paths, equal to the cyclomatic complexity. The algorithm guarantees full path coverage Li et al. [12], proposed an Ant Colony Optimization technique for automatic generation of test sequences for state based software testing. The techniques illustrated that it can directly use an UML diagram to produce test sequences to achieve required test coverage.

The effectiveness of testing can be attained by state transition testing and path based testing. State transition testing and path based testing are normally done for functional as well as structural testing of system software. Bhuvnesh et al. [13] proposed an algorithm for generation of test sequences from the state transition of the software and also path generation from th8e control flow graph of the software using the basic property and behaviour of ants.

S. D. Shtovba [15], reviewed the theory and applications of ant algorithms, new methods of discrete optimization based on the simulation of self-organized colony of biologic ants. He stated the principle of ACO and demonstrated the self-organization principles of social insects and explains the way the ants locate the shortest path. He used travelling salesperson as an example and applied ACO with it to compare its result

Chapter 4

Proposed Work

Our project consists of some existing work as well as some proposed work. Chandler and others [10] gave an insight about how to extract nodes from UML model without using any intermediate model. XMI representation of the UML model file is used to extract information about the diagram. Praveen et al. [6] proposed a technique to generate test sequences from UML activity diagram using control flow graph as an intermediate model. But in our approach, we generated test sequences from the diagram using same algorithm proposed by him but without generating any intermediate model. We used XMI representation from the corresponding activity diagram.

4.1 Generation of test sequences from Activity diagram

In UML-based testing, type of the diagram decides the testing approach. However, test sequence generation from these diagram is mainly focused on the intermediate representation to which these diagrams are transformed. Here we focus on only Activity diagram of the software. Therefore, some of the testing are graph-based testing, direct UML specification testing.

In graph-based testing, UML diagram is transformed into an intermediate graph. From that graph, corresponding path will be generated using either DFS or BFS or any other traversing algorithm. A path in a graph is a sequence of vertices and from each of its vertices there is an edge to the next vertices in sequence. From testing prospective, it is important to search paths in a graph representation of the UML model, and test data values that solves constraints along the paths.

Some approaches process UML specifications without generating any intermediate model. These methods encourage for direct processing of UML specifications available from visual diagram drawing tools. In recent years, Object Management Group (OMG) defined a standard for exchanging metadata information through Extensible Markup Language (XML) called XML Metadata Interchange (XMI). Direct processing of UML model has become simple due to XMI representation of UML models and parsers like DOM and SAX.

In our project,

- We have used IBM Rational Software Architect (RSA) to model the activity diagram for a software.
- We have used Direct UML Specification processing technique to generate the XMI representation from the UML Model.
- SAX parser to parse the XMI representation to collect corresponding nodes and edges.
- After using a traversal algorithm on those generated nodes and corresponding edges between them we have generated the test sequences.

Steps for Test Sequence generation:

1. First an activity diagram is taken.
2. XML file is extracted from the diagram and taken as an input to the program.
3. Nodes and Edges are extracted from the XMI representation and stored into a linked list.
4. Source and Destination of each edge are extracted and stored in linked lists.
5. Decision nodes, Fork nodes and join nodes are stored in linked lists
6. DFS algorithm is applied to generate test sequences.

Example

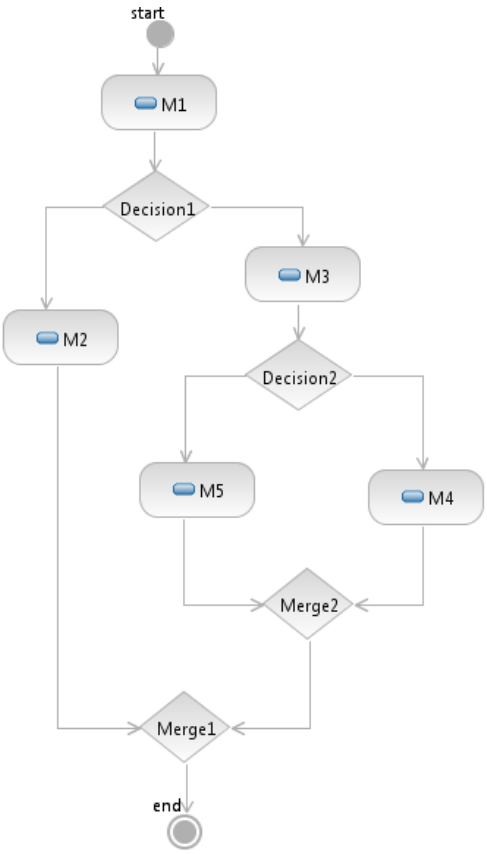
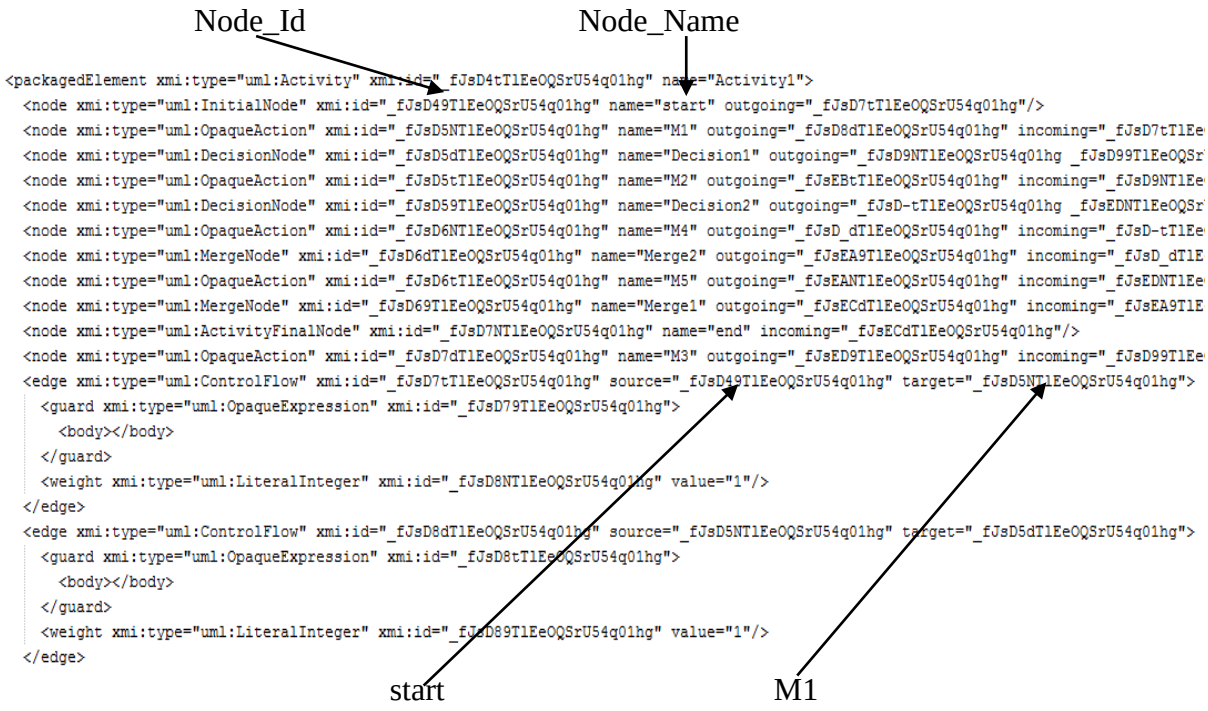


Figure 4.1 Sample activity diagram

XMI representation of this diagram is extracted and stored in a file. From there, nodes and edges are extracted and stored in an array or linked list. Using this information we can draw graph or tree out of it using other tools. Using some traversing algorithm we can generate required test sequences from it.




```
<packagedElement xmi:type="uml:Activity" xmi:id="_fJsD4tTlEeOQSrU54q01hg" name="Activity1">
  <node xmi:type="uml:InitialNode" xmi:id="_fJsD49TlEeOQSrU54q01hg" name="start" outgoing="_fJsD7tTlEeOQSrU54q01hg"/>
  <node xmi:type="uml:OpaqueAction" xmi:id="_fJsD5NTlEeOQSrU54q01hg" name="M1" outgoing="_fJsD8dTlEeOQSrU54q01hg" incoming="_fJsD7tTlEe
  <node xmi:type="uml:DecisionNode" xmi:id="_fJsD5dTlEeOQSrU54q01hg" name="Decision1" outgoing="_fJsD9NTlEeOQSrU54q01hg _fJsD99TlEeOQSr
  <node xmi:type="uml:OpaqueAction" xmi:id="_fJsD5tTlEeOQSrU54q01hg" name="M2" outgoing="_fJsEBtTlEeOQSrU54q01hg" incoming="_fJsD9NTlEe
  <node xmi:type="uml:DecisionNode" xmi:id="_fJsD59TlEeOQSrU54q01hg" name="Decision2" outgoing="_fJsD-tTlEeOQSrU54q01hg _fJsEDNTlEeOQSr
  <node xmi:type="uml:OpaqueAction" xmi:id="_fJsD6NTlEeOQSrU54q01hg" name="M4" outgoing="_fJsD_dTlEeOQSrU54q01hg" incoming="_fJsD-tTlEe
  <node xmi:type="uml:MergeNode" xmi:id="_fJsD6dTlEeOQSrU54q01hg" name="Merge2" outgoing="_fJsEA9TlEeOQSrU54q01hg" incoming="_fJsD_dTlE
  <node xmi:type="uml:OpaqueAction" xmi:id="_fJsD6tTlEeOQSrU54q01hg" name="M5" outgoing="_fJsEANTlEeOQSrU54q01hg" incoming="_fJsEDNTlEe
  <node xmi:type="uml:MergeNode" xmi:id="_fJsD69TlEeOQSrU54q01hg" name="Merge1" outgoing="_fJsECdTlEeOQSrU54q01hg" incoming="_fJsEA9TlE
  <node xmi:type="uml:ActivityFinalNode" xmi:id="_fJsD7NTlEeOQSrU54q01hg" name="end" incoming="_fJsECdTlEeOQSrU54q01hg"/>
  <node xmi:type="uml:OpaqueAction" xmi:id="_fJsD7dTlEeOQSrU54q01hg" name="M3" outgoing="_fJsED9TlEeOQSrU54q01hg" incoming="_fJsD99TlEe
  <edge xmi:type="uml:ControlFlow" xmi:id="_fJsD7tTlEeOQSrU54q01hg" source="_fJsD49TlEeOQSrU54q01hg" target="_fJsD5NTlEeOQSrU54q01hg">
    <guard xmi:type="uml:OpaqueExpression" xmi:id="_fJsD79TlEeOQSrU54q01hg">
      <body></body>
    </guard>
    <weight xmi:type="uml:LiteralInteger" xmi:id="_fJsD8NTlEeOQSrU54q01hg" value="1"/>
  </edge>
  <edge xmi:type="uml:ControlFlow" xmi:id="_fJsD8dTlEeOQSrU54q01hg" source="_fJsD5NTlEeOQSrU54q01hg" target="_fJsD5dTlEeOQSrU54q01hg">
    <guard xmi:type="uml:OpaqueExpression" xmi:id="_fJsD8tTlEeOQSrU54q01hg">
      <body></body>
    </guard>
    <weight xmi:type="uml:LiteralInteger" xmi:id="_fJsD89TlEeOQSrU54q01hg" value="1"/>
  </edge>
```

Figure 4.2 XMI representation of the diagram given in Figure 4.1

Using the DOM or SAX parser, we can take this XMI representation file as an input to the program and extract necessary information out of it. In this way, node_id and node_name are stored in linked list. Similarly edges containing source and destination are stored in linked list. Using DFS traversal technique, we can generate test sequences of the diagram.

Table 4.1 Generated Test Sequences from the activity diagram given in Figure 4.1

Sl No.	Test Sequence
1	start, M1, Decision1, M3, Decision2, M5, Merge2, Merge1, end
2	start, M1, Decision1, M3, Decision2, M4, Merge2, Merge1, end
3	start, M1, Decision1, M2, Merge1, end

4.2 Prioritisation

Test sequence generation and selection of the most suitable one is the most important task in software testing because testing a software using an effective one can save more time and cost than testing with any other test sequences. Therefore selecting the right test sequence is one of the top software testing technique. Due to time, cost and other criteria manual testing is not possible and also not useful, therefore there is a need for automated testing. Producing automated and effective test sequences is a very difficult task in the software testing process. Prioritized test sequence can reduce the overall cast of testing as well as increases the chance to find fault in the software more effectively.

4.2.1 Weight assignment

Basically in the activity diagram, Decision nodes and Fork nodes can be considered as important module of the software. So a test sequence having maximum number of decision and fork nodes can be considered for testing purpose. That is why in this approach, some weight to decision nodes and fork nodes have been assigned so that the test sequence containing more of those will be considered as a prioritized path. So for prioritization following steps are followed.

- Calculate FAN-IN and FAN-OUT of each node
  FAN-IN(i) : Number of incoming edges to node i

  FAN-OUT(i): Number of outgoing edges from node i

  Weight of node i = FAN-IN(i) * FAN-OUT(i)

- Assign highest priorities to decision and fork nodes (e.g. weights 5 and 10)

Example

Using the example from Figure 4.1,

Weight of the each individual node i = weight of node i

If node i is a decision or fork node then = weight of node i + weight of decision or fork node

Table 4.2 Nodes and corresponding weights from the diagram given in Figure 4.1

Node_name	Weight
Start	0
M1	1
Decision1	7
M2	1
Decision2	7
M4	1
Merge2	2
M5	1
Merge1	2
End	0
M3	1

Table 4.3 Test sequences generated as well as corresponding strength

Sl No.	Test Sequence	Strength
1	start, M1, Decision1, M3, Decision2, M5, Merge2, Merge1, end	21
2	start, M1, Decision1, M3, Decision2, M4, Merge2, Merge1, end	21
3	start, M1, Decision1, M2, Merge1, end	11

From the current approach there are some disadvantages, one of them was it extracts all possible path from the diagram and prioritizes them. But generating all sequences will consume more time. So a heuristic approach is required to limit number of sequences as well as gives us the best result out of it. So Ant Colony Optimization is taken into consideration.

4.2.2 Ant Colony Optimization

There is no test sequence which can ensure detection of all defects. So main part is how to select test sequence that can increases the probability of finding defects. Tester's objective is to test all possible sequence which he/she identified from the diagram, but generating all the paths for testing is not a suitable technique. So total number of test sequences using cyclomatic complexity can be easily calculated, but the problem is how to select optimal path.

In this type of requirement, Ant's behaviour is useful. Blind Ants coordinate their activities via stigmergy [6]. The most important thought is that the self-organizing principles which allow highly coordinated behaviour of real ants can be used with artificial agents which collaborate to solve the complex computational problems. In reality, ants coordinate with each other by sensing the pheromone level on the path. Selection of path by ants depends on probability theory. Ants generates pheromone on the path each time it traverses. So it can be useful for sensing the path for future references. By understanding the pheromone level as well as the heuristic knowledge of each path, ant can choose their path. So a path having probability of highest pheromone and heuristic knowledge is more likely to be selected.

Path generation using Ant Colony Optimization:

Ant has ability to collect the knowledge of all feasible paths from its current state.

- Ant has three information about the path:
    - Pheromone of the path (p) - It is the pheromone trail that ant leaves behind when it traverses an edge. It helps other ants to make decision in the future.

- Heuristic (h) - Which indicates the visibility of a path for an ant at the current vertex.
- Visited status (v) - Helps ant to determine whether the node has been visited earlier or not.

Algorithm:

1. Set pheromone value for each edges to 1.

   Set heuristic value for each edges to 2.

   Set Visited status for every node to 0.

   Set sum = 0.

   Set current node = "start".

   count = Number of edges – Number of nodes + 2 (Cyclomatic Complexity)

2. Traverse from the current node(i)

   Update visited node status for current node from 0 to 1.

3. If the current node is Decision node

   if 'i' is the decision node and 'j' and 'k' are nodes connected to it.

   Find probability of path i->j and i->k

$$\text{Pro(ij)} = \frac{p(ij)*h(ij)^{-1}}{p(ij)*h(ij)^{-1}+p(ik)*h(ik)^{-1}} \quad \ldots(1)$$

$$\text{Pro(ik)} = \frac{p(ik)*h(ik)^{-1}}{p(ij)*h(ij)^{-1}+p(ik)*h(ik)^{-1}} \quad \ldots(2)$$

   3.1 If Pro(ij) > Pro(ik) then select i->j path

   3.2 If Pro(ij) < Pro(ik) then select i->k path

   3.3 If Pro(ij) = Pro(ik) then

   3.3.1 if(Connected node to current node is "end")

then set next node = “end”

3.3.2 if(v[j] != v[k])

if(v[j]==0) then choose i->j path

if(v[k]==0) then choose i->k path

3.3.3 if(v[j] == v[k])

then choose randomly

4. Update Pheromone of the selected path i->j

$p(ij) = p(ij) + h(ij)^{-1}$

$h(ij) = 2*h(ij)$

5. Sum = Sum + p(ij)

Strength[count] = sum

set current node = next_vertex.

6. If current node != “end”, Display the path and Strength[count]. go to step 2

7. Set count = count – 1

8. If count>0

set visit status(v) of all node to 0, sum = 0, set current node = “start” and go to step 2

9. End of algorithm

Example

Table 4.4 Test Sequences generated and their corresponding strength given in Figure 4.1

Sl No.	Test Sequence	Strength
1	start, M1, Decision1, M3, Decision2, M4, Merge2, Merge1, end	12
2	start, M1, Decision1, M2, Merge1, end	8.25
3	start, M1, Decision1, M3, Decision2, M5, Merge2, Merge1, end	13.87

4.2.3 Proposed test sequence prioritization technique using ACO

The previous approach gives us the test sequences according to the length of the path. Test sequence having higher length are given higher priority. But the concept adopted earlier was, test sequence having maximum number of decision nodes and fork node are given higher priority. As the previous algorithm is a path based testing it does not depend on number of decision nodes and fork nodes. Also that does not work for the diagram containing fork nodes. So that algorithm need to be modified on the basis of weightage of nodes.

So the following algorithm has some modifications,

- Add fork and join to the algorithm.
- Give priority to nodes instead of edges. Set pheromone and heuristics values of node instead of edges. So that it will be easier to add weights to decisions and forks.
- Give weight of Decision = Total number of nodes

  weight of fork = Total number of nodes + 5
- Calculate FANIN and FANOUT of each node

  FANIN(i) : Number of incoming edges to a node i

  FANOUT(i): Number of outgoing edges to node I

  TOTAL(i) = FANIN(i) * FANOUT(i) : weight of the node I

Here path will be generated according to the ant colony optimization technique but nodes will be given priority instead of edges.

Table 4.5 Test Sequences and corresponding Strength

Sl No.	Test Sequence	Strength
1	start, M1, Decision1, M3, Decision2, M5, Merge2, Merge1, end	42.0
2	start, M1, Decision1, M2, Merge1, end	23.75
3	start, M1, Decision1, M3, Decision2, M4, Merge2, Merge1, end	43.81

Here, test sequence 1 and 3 are considered to be most preferred as it contains more number of decision nodes. Test sequence 3 has more strength than test sequence 1 even if they both have same number of nodes. This is because every time an ant traverses along a path it updates corresponding heuristic and pheromone value of each edges. So when 3rd ant goes through a path, pheromone value of each edge is higher than the time when 1st ant went. Also the strength of the path depends upon the pheromone value of the edges encountered in that path. So the strength of the 3rd test sequence is higher than the strength of the 1st test sequence.

Chapter 5

Implementations and Results

Here we implement all algorithms proposed earlier with java program. We take XMI representation of the activity diagram as an input to the program and following the respective algorithm we generate desired test sequences along with their strength.

5.1 Tools used

We use the following tools in order to implement and code the programs and finally to get the desired test sequences.

• Eclipse

• RSA

• Graphviz

5.1.1 Eclipse

It is a multi-language software development environment constituting of an integrated development environment and an extensible plug-in system. It is written primarily in Java and can be used to develop applications in Java and, but it can be used for other languages by means of the various plug-ins. The other languages include C, C++, COBOL, Python, Perl, PHP, and others. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C, Eclipse JDT for Java and Eclipse PDT for PHP. The most important feature of Eclipse is its plug-in system. We can integrate different plug-in tools into the eclipse environment and can be used them in the applications. It is also simple to use as we need not install it. The only thing that

we have to do it is to download Eclipse and run the eclipse.exe file. .We have to download and install MinGW GCC compiler for compilation of C++ code.

5.1.2 RSA

RSA stands for Rational Software Architect. It is a product from IBM. It is a modeling and development environment that uses the Unified Modeling Language (UML) for designing architecture for C++ and Java 2 Enterprise Edition (J2EE) applications and web services. We can design many types of structural as well as behavioural diagram in it. From it we can export the XMI representation of the respective diagram and use it in java programming for parsing.

5.1.3 Graphviz

Graphviz is a tool that can pictorially represent a graph. We have used this tool in our project to visualize our final output in a better way i.e. in the form of a pictorial graph instead of an adjacency matrix or adjacency list. The output of the program is converted into a form that is recognizable by graphviz and is written into an output file in the same format. Graphviz reads from the output file in order to generate the graph that the user can visualize.

5.2 Screenshots

5.2.1 Generation of test sequence From Activity diagram

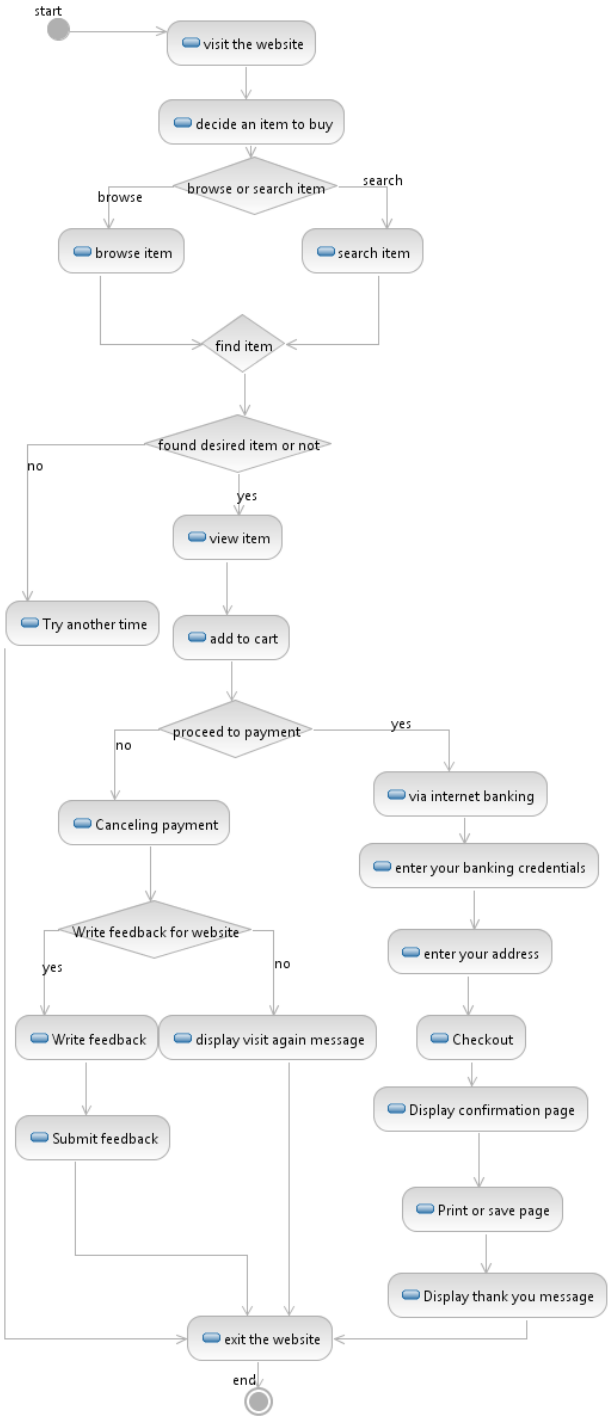


Figure 5.1 Activity Diagram of Online shopping

Figure 5.2 XMI representation of the activity diagram given in Figure 5.1

Using the SAX parser, nodes and edges from the corresponding XMI representation is extracted and saved in linked lists. Source and destination of each edges are extracted and saved in linked list.

Two linked lists are made, Node_name and Node_id. Another two linked list are made, Source of an edge and destination of an edge. Initially source and destination of each edges contained Node_id of nodes as from the XMI representation (Figure 5.2). After comparing those node_ids with corresponding Node_names we changed the source and edges of each edges respectively. Tabled view of Nodes and Edges are given in next page.

Table 5.1 Node_names and their corresponding Node_ids

Node Name	Node Id
start	_yky-o9UaEeO18ZGvLO-emQ
visit the website	_yky-pNUaEeO18ZGvLO-emQ
browse or search item	_yky-pdUaEeO18ZGvLO-emQ
browse item	_yky-ptUaEeO18ZGvLO-emQ
search item	_yky-p9UaEeO18ZGvLO-emQ
found desired item or not	_yky-qNUaEeO18ZGvLO-emQ
find item	_yky-qdUaEeO18ZGvLO-emQ
decide an item to buy	_yky-qtUaEeO18ZGvLO-emQ
view item	_yky-q9UaEeO18ZGvLO-emQ
add to cart	_yky-rNUaEeO18ZGvLO-emQ
proceed to payment	_yky-rdUaEeO18ZGvLO-emQ
via internet banking	_yky-rtUaEeO18ZGvLO-emQ
enter your banking credentials	_yky-r9UaEeO18ZGvLO-emQ
enter your address	_yky-sNUaEeO18ZGvLO-emQ
end	_yky-sdUaEeO18ZGvLO-emQ
exit the website	_ykzlsNUaEeO18ZGvLO-emQ
Cancelling payment	_ykzlsdUaEeO18ZGvLO-emQ
Try another time	_ykzlstUaEeO18ZGvLO-emQ
Checkout	_ykzls9UaEeO18ZGvLO-emQ
Display confirmation page	_ykzltNUaEeO18ZGvLO-emQ
Print or save page	_ykzltdUaEeO18ZGvLO-emQ
Display thank you message	_ykzlttUaEeO18ZGvLO-emQ
Write feedback for website	_ykzlt9UaEeO18ZGvLO-emQ
Write feedback	_ykzluNUaEeO18ZGvLO-emQ
Submit feedback	_ykzludUaEeO18ZGvLO-emQ
display visit again message	_ykzlutUaEeO18ZGvLO-emQ

This is a table of node name and node id extracted from the XMI representation code (Figure.5.2). Each node has a unique node_id, so they can be distinguished from each other by their node_id. Similarly, each edge has a unique node_id for source and target section which can be seen from the Figure 5.2.

Table 5.2 Source and Destination of each edge

Source of Edges	Destination of Edges
browse or search item	browse item
browse or search item	search item
start	visit the website
visit the website	decide an item to buy
search item	find item
browse item	find item
find item	found desired item or not
decide an item to buy	browse or search item
found desired item or not	view item
view item	add to cart
add to cart	proceed to payment
via internet banking	enter your banking credentials
enter your banking credentials	enter your address
proceed to payment	Cancelling payment
found desired item or not	Try another time
exit the website	end
Cancelling payment	Write feedback for website
Try another time	exit the website
enter your address	Checkout
proceed to payment	via internet banking
Checkout	Display confirmation page
Display confirmation page	Print or save page
Print or save page	Display thank you message
Display thank you message	exit the website
Write feedback for website	Write feedback
Write feedback	Submit feedback
Write feedback for website	display visit again message
Submit feedback	exit the website
display visit again message	exit the website

From the list of edges and nodes, we can easily traverse the Activity diagram using any traversal technique.

5.2.2 Prioritization

Prioritization of test sequences are done using proposed and existing algorithms. UML activity diagram is taken into example and generated test sequences are prioritized according to their strength calculated using algorithms. Test sequence having higher strength is the most suitable one for testing. Here activity diagram of online shopping is taken into consideration (Figure 5.1). Here fork node is not taken, as ACO algorithm does not consider fork nodes. So for simplicity one diagram (Figure 5.1) is taken to compare all approach.

5.2.2.1 Weight Assignment

Traversing the nodes using DFS algorithm, we can generate the test sequences and their corresponding strength.

```
Test Sequence 1
[start, visit the website, decide an item to buy, browse or search item, search item, find item, found desired item
or not, Try another time, exit the website, end] no. of nodes 10 strength: 24  No of decision:2 No of Fork:0

Test Sequence 2
[start, visit the website, decide an item to buy, browse or search item, search item, find item, found desired item
or not, view item, add to cart, proceed to payment, via internet banking, enter your banking credentials, enter your
address, Checkout, Display confirmation page, Print or save page, Display thank you message, exit the website, end]
no. of nodes 19 strength: 39 No of decision:3 No of Fork:0

Test Sequence 3
[start, visit the website, decide an item to buy, browse or search item, search item, find item, found desired item
or not, view item, add to cart, proceed to payment, Cancelling payment, Write feedback for website, display visit
again message, exit the website, end] no. of nodes 15 strength: 41 No of decision:4 No of Fork:0

Test Sequence 4
[start, visit the website, decide an item to buy, browse or search item, search item, find item, found desired item
or not, view item, add to cart, proceed to payment, Cancelling payment, Write feedback for website, Write feedback,
Submit feedback, exit the website, end] no. of nodes 16 strength: 42 No of decision:4 No of Fork:0

Test Sequence 5
[start, visit the website, decide an item to buy, browse or search item, browse item, find item, found desired item
or not, Try another time, exit the website, end] no. of nodes 10 strength: 24 No of decision:2 No of Fork:0

Test Sequence 6
[start, visit the website, decide an item to buy, browse or search item, browse item, find item, found desired item
or not, view item, add to cart, proceed to payment, via internet banking, enter your banking credentials, enter your
address, Checkout, Display confirmation page, Print or save page, Display thank you message, exit the website, end]
no. of nodes 19 strength: 39 No of decision:3 No of Fork:0

Test Sequence 7
[start, visit the website, decide an item to buy, browse or search item, browse item, find item, found desired item
or not, view item, add to cart, proceed to payment, Cancelling payment, Write feedback for website, display visit
again message, exit the website, end] no. of nodes 15 strength: 41 No of decision:4 No of Fork:0

Test Sequence 8
[start, visit the website, decide an item to buy, browse or search item, browse item, find item, found desired item
or not, view item, add to cart, proceed to payment, Cancelling payment, Write feedback for website, Write feedback,
Submit feedback, exit the website, end] no. of nodes 16 strength: 42 No of decision:4 No of Fork:0
```

Figure 5.3 Generated test sequences from activity diagram given in Figure 5.1

The activity diagram is parsed through the program and used DFS algorithm on the extracted informations. The output contains test sequences along with the total number of numbers, decision nodes as well as fork nodes.

Table 5.3 Test sequences, Strength, Number of nodes, decision nodes given in Figure 5.3

Test Sequence	**Strength**	**No. of Nodes**	**No. of decisions**
Test Sequence 4	42	16	4
Test Sequence 8	42	16	4
Test Sequence 3	41	15	4
Test Sequence 7	41	15	4
Test Sequence 2	39	19	3
Test Sequence 6	39	19	3
Test Sequence 1	24	10	2
Test Sequence 5	24	10	2

From Table 5.3, we got the test sequences according to the number of decision nodes. But the drawback of DFS algorithm implementation is that, it extracts all possible test sequences from a model. But in real life there may be a lot of test sequences for a single model. So checking all the test sequences will consume more time. If there are N decision nodes, then there can be 2^N test sequences. So to reduce the number of the number of test sequences Cyclomatic complexity is introduced. Cyclomatic complexity determines the number of linearly independent test sequences. Linearly independent test sequence is a test sequence that have at least one new node.

5.2.2.2 Ant Colony Optimization

Here cyclomatic complexity is used, and according to it test sequences are generated. Using the algorithm, test sequences are generated from the activity diagram given in Figure5.1,

```
Test Sequence 1
[start, visit the website, decide an item to buy, browse or search item, browse item,
find item, found desired item or not, Try another time, exit the website, end]
Strength: 13.5 no of nodes: 10 no of fork: 0 no of decision: 2

Test Sequence 2
[start, visit the website, decide an item to buy, browse or search item, search item,
find item, found desired item or not, view item, add to cart, proceed to payment, via
internet banking, enter your banking credentials, enter your address, Checkout,
Display confirmation page, Print or save page, Display thank you message, exit the
website, end] Strength: 28.25 no of nodes: 19 no of fork: 0 no of decision: 3

Test Sequence 3
[start, visit the website, decide an item to buy, browse or search item, search item,
find item, found desired item or not, Try another time, exit the website, end]
Strength: 16.375 no of nodes: 10 no of fork: 0 no of decision: 2

Test Sequence 4
[start, visit the website, decide an item to buy, browse or search item, browse item,
find item, found desired item or not, view item, add to cart, proceed to payment,
Cancelling payment, Write feedback for website, display visit again message, exit the
website, end] Strength: 24.4375 no of nodes: 15 no of fork: 0 no of decision: 4

Test Sequence 5
[start, visit the website, decide an item to buy, browse or search item, search item,
find item, found desired item or not, view item, add to cart, proceed to payment,
Cancelling payment, Write feedback for website, Write feedback, Submit feedback, exit
the website, end] Strength: 27.296875 no of nodes: 16 no of fork: 0 no of decision: 4
```

Figure 5.4 Test sequences generated from Figure 5.1 using ACO

Here, cyclomatic complexity according to the formula is calculated as 5. So total number of test sequences generated is 5. Thus it generates less number of test sequences than the previous approach.

Table 5.4 Test sequences, Strength, Number of nodes, decision nodes given in Figure 5.4

Test Sequence	**Strength**	**No of Nodes**	**No of decisions**
Test Sequence 2	28.15	19	3
Test Sequence 5	27.29	16	4
Test Sequence 4	24.43	15	4
Test Sequence 3	16.37	10	2
Test Sequence 1	13.5	10	2

From the table (Table. 5.4) it is found that the test sequence having higher number of nodes is given higher priority. It does not depend on the number of decision nodes. As it is a path based testing [6], the algorithm also generates the test sequence having higher number of

nodes. The algorithm is proposed in keeping in mind that path based testing is the most suitable testing. But if a software has different modules, then test sequence should be chosen in such a way that the number of modules it encounters is highest from other. If a module has a very long sequence of action and during path based testing if it generates the test sequence having the nodes of that module only then it will not be the efficient test sequence for testing. We need test sequence having maximum number of modules in it. In this case, we need a sequence having maximum number of decision node.

This approach generates test sequences according to the size of the path. Test path having higher length are given higher priority. But using the black box testing, we want to test the functionality of the software not internal working or completeness of the software. As this algorithm is a path based testing it does not depend on number of decision nodes and fork nodes. Also this algorithm does not work for the diagram containing fork nodes. Here we can see Figure 5.1 does not have fork nodes. So this algorithm need to be modified on the basis of weightage of nodes.

5.2.2.3 Proposed test sequence prioritization Technique using ACO:

Using the same figure Figure 5.1 for test sequence generation,

```
Test Sequence 1
[start, visit the website, decide an item to buy, browse or search item, browse item,
find item, found desired item or not, view item, add to cart, proceed to payment, via
internet banking, enter your banking credentials, enter your address, Checkout,
Display confirmation page, Print or save page, Display thank you message, exit the
website, end] Strength: 124.5 no of nodes: 19 no of fork: 0 no of decision: 3

Test Sequence 2
[start, visit the website, decide an item to buy, browse or search item, search item,
find item, found desired item or not, Try another time, exit the website, end]
Strength: 77.75 no of nodes: 10 no of fork: 0 no of decision: 2

Test Sequence 3
[start, visit the website, decide an item to buy, browse or search item, browse item,
find item, found desired item or not, Try another time, exit the website, end]
Strength: 78.875 no of nodes: 10 no of fork: 0 no of decision: 2

Test Sequence 4
[start, visit the website, decide an item to buy, browse or search item, search item,
find item, found desired item or not, view item, add to cart, proceed to payment,
Cancelling payment, Write feedback for website, display visit again message, exit the
website, end] Strength: 142.9375 no of nodes: 15 no of fork: 0 no of decision: 4

Test Sequence 5
[start, visit the website, decide an item to buy, browse or search item, browse item,
find item, found desired item or not, view item, add to cart, proceed to payment,
Cancelling payment, Write feedback for website, Write feedback, Submit feedback, exit
the website, end] Strength: 146.54297 no of nodes: 16 no of fork: 0 no of decision: 4
```

Figure 5.5 Test sequences generated from the activity diagram given in Figure 5.1 using the proposed algorithm

As it can be seen test sequence 5 has highest priority as it contains all the decision nodes and fork nodes of the diagram. The strength of the test sequence comes from the pheromone value of the node as well as the weights of each nodes. Here test sequence 5 gives maximum strength as it contains highest number of decision nodes. Although test sequence 4 and test sequence 5 both have same number of decision nodes, Test sequence 5 is given more priority as total number of nodes in test sequence 5 is more than test sequence 4. So here ACO algorithm also applies. This algorithm emphasizes both on the number of the nodes as well as the number of decision nodes.

Table 5.5 Test sequences, Strength, Number of nodes, decision nodes given in Figure 5.5

Test Sequence	**Strength**	**No of Nodes**	**No of decisions**
Test Sequence 5	146.45	16	4
Test Sequence 4	142.93	15	4
Test Sequence 1	124.5	19	3
Test Sequence 3	78.87	10	2
Test Sequence 2	77.75	10	2

From above table, we found out that those having greater number of decision nodes are given higher priorities. It also limits the redundant test sequences so it gives the desired result.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have been able to made intermediate representation which known as XMI representation from the UML diagram. This type of intermediate representation process can also be referred as direct UML specification. From this XMI representation, we are able to extract corresponding nodes and edges between them. Using those we have generated test sequences on using a traversal algorithm. On applying different prioritization techniques we are able to generate test sequences on the basis of priority. From that we can use test sequence having highest priority level for testing purpose.

6.2 Future Work

Different UML diagrams will be taken as input to the program and its test sequences will be generated. Different prioritization techniques will be implemented and reviewed. We will implement comparison between two different activity diagram and its similarity will be given in a graph.

Test sequence will be generated using different intermediate model-based testing such as graph-based testing.

Chapter 7

Reference

1. Sangeeta Sabharwal, Ritu Sibal and Chayanika Sharma. "Applying Genetic Algorithm for Prioritization of Test Case Scenarios Derived from UML Diagrams" IJCSI pages 433-444, 2011
2. Mahesh Shirole and Rajeev Kumar. "Testing for Concurrency in UML Diagrams" ACM SIGSOFT, Volume 37 page 1-8, 2012
3. Ian Sommerville," Software Engineering", eight Edition, Pearson Edition, 2009.
4. Phil. McMinn, "Search-Based Software Test Data Generation: A Survey", Software Testing, Verification and Reliability, Vol 14, No 3, pp. 212-223, 2004.
5. Cormen, Leiserson, Rivest and Stein. "Introduction to Algorithms", Third Edition, 2012.
6. Praveen Ranjan Srivastava, Km Baby and G Raghurama. "An Approach of Optimal Path Generation using Ant Colony Optimization", TENCON 2009
7. Aditya P. Mathur "Foundation of Software Testing", First Edition Pearson Education, 2007.
8. W. E. Howden, "Functional program Testing and Analysis", McGrawHill, 1987.
9. Dorigo M., Maniezzo, V., Colorni, A., "Positive Feedback as a Search Strategy", Technical Report No. 91016, Politecnico di Milano, Italy, 1991.
10. R. Chandler, C. P. Lam, and H. Li. AD2US: An Automated Approach to Generating Usage Scenarios from UML Activity Diagrams. In Proceedings of the 12th Asia-Pacific Software Engineering Conference, pages 9-16. IEEE Computer Society, 2005.

11. C. Sun, B. Zhang, and J. Li. TSGen: A UML Activity Diagram-Based Test Scenario Generation Tool. In Proceedings of the 2009 International Conference on Computational Science and Engineering - Volume 02, CSE '09, pages 853-858, Washington, DC, USA, 2009.
12. Huaizhong LI and C. Peng LAM, "An Ant Colony Optimization Approach to Test Sequence Generation for Statebased Software Testing", Proceedings of the Fifth International Conference on Quality Software (QSIC'05) , IEEE, 1550-6002/05.
13. Bhuvnesh Sharma, Isha Girdhar, Monika Taneja, Pooja Basia, Sangeetha Vadla, Praveen Ranjan Srivastava, "Software Coverage : A Testing Approach through Ant Colony Optimization" SEMCCO 2011, part 1, LNCS 7076, pp 618-625, 2011 Springer-Verlag Berlin Heidelberg, 2011
14. U. Farooq, C. P. Lam and H. Li, "Towards Automated Test Sequence Generation" IEEE, 2008, 1530-0803/08.